IoT Communication Model

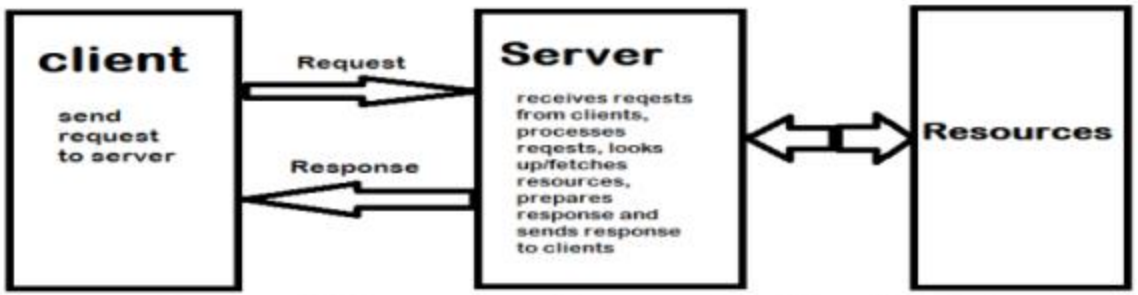
(Just for knowledge and concept not in our syllabus)

IoT Communication Model

- Request-Response
- Publish-Subscribe
- Push-Pull
- Exclusive Pair

Request-Response

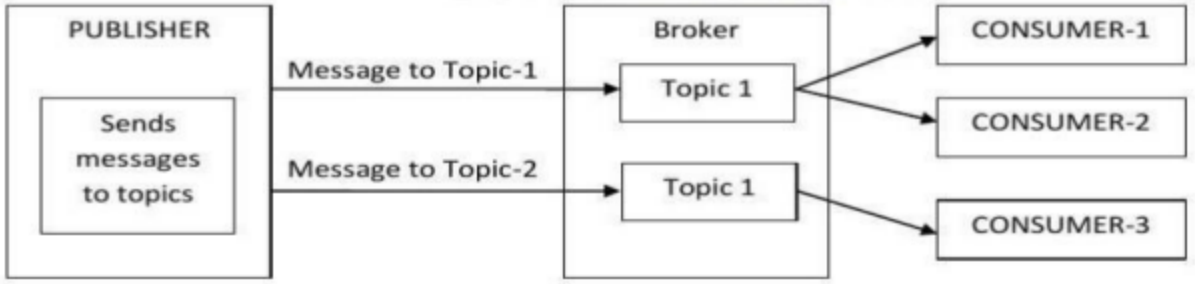
- In Request–Response communication model client sends request to the server and the server responds to the request.
- When the server receives the request it decides how to respond, fetches the data, retrieves resources, and prepares the response and sends to the client.
- R-R is a communication model where request and response pair is independent of each others.



Request-Response Communication Model

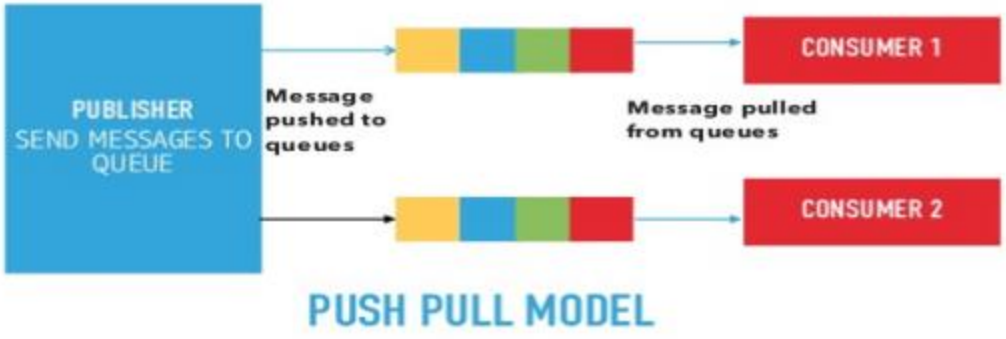
Publish-Subscribe:

- This model involves publishers, brokers and consumers. Publishers are the sources of data.
- Publishers send the data to the topic which are managed by the broker. Publishers are not aware of the consumers.
- Consumers subscribe to the topics which are managed by the broker.
- When broker receives the data from the Publisher, it sends to all the consumers.



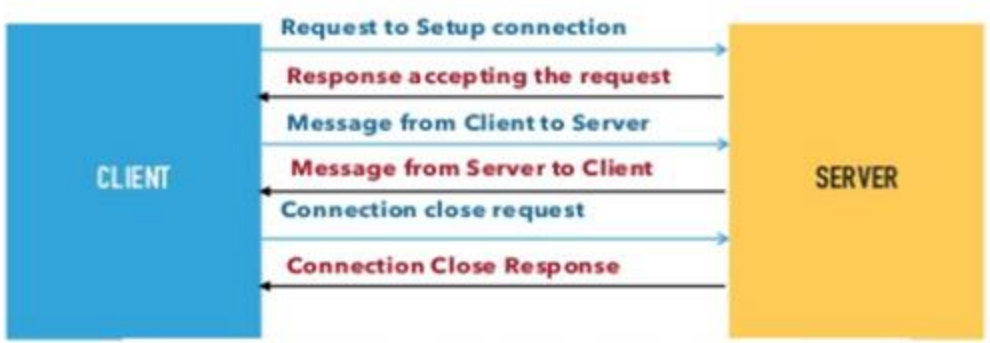
Push-Pull

- In this model the producers push the data in queues and the consumers pull the data from the queues.
- Producers do not need to be aware of the consumers.
- Queues help in decoupling the messaging between the producer and consumers.
- Queues also act as buffer which helps in situation when there is mismatch between the rate at which the producers push the data and consumers pull the data.



Exclusive Pair:

- Exclusive pair is a bi-directional, fully duplex communication model that uses a persistent connection between the client and server, once the connection is established it remains open until the client sends a request to closer the connection.
- Client and server can send the message to each other after connection setup.
- In this model server is aware of all the open connection.



EXCLUSIVE PAIR COMMUNICATION MODEL

IOT Communication API's

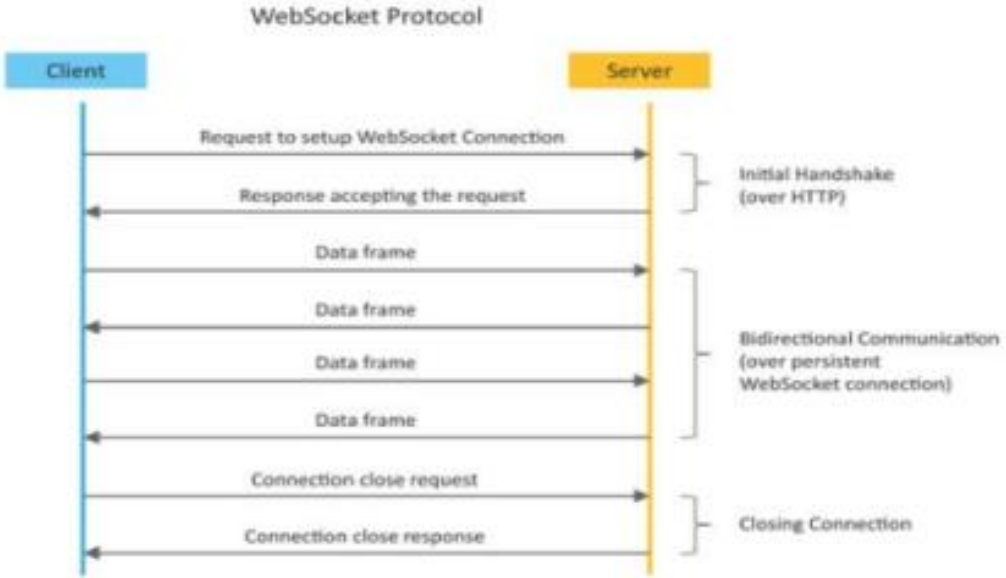
- The application program (or programming) interface, or API, that really ties together the connected "things" of the "internet of things."
- IoT APIs are the points of interaction between an IoT device and the internet and/or other elements within the network.
- Generally we used Two APIs For IoT Communication. These IoT Communication APIs are:
  - REST-based Communication APIs
  - WebSocket-based Communication APIs

1. REST-based Communication APIs

- Representational state transfer (REST) is a set of architectural principles by which you can design Web services the Web APIs that focus on systems resources and how resource states are addressed and transferred.
- REST is most popular IoT Communication APIs.
- REST APIs that follow the request response communication model.

2. WebSocket based communication API

- Websocket APIs allow bi-directional, full duplex communication between clients and servers.
- Websocket APIs follow the exclusive pair communication model.
- Unlike request-response model such as REST, the WebSocket APIs allow full duplex communication and do not require new connection to be setup for each message to be sent.
- Websocket communication begins with a connection setup request sent by the client to the server.
- The request (called websocket handshake) is sent over HTTP and the server interprets it is an upgrade request.
- If the server supports websocket protocol, the server responds to the websocket handshake response.
- After the connection setup client and server can send data/mesages to each other in full duplex mode.
- Websocket API reduce the network traffic and letency as there is no overhead for connection setup and termination requests for each message.
- Websocket suitable for IoT applications that have low latency or high throughput requirements.
- So Web socket is most suitable IoT Communication APIs for IoT System.


WebSocket Protocol
Client
Server
Request to setup WebSocket Connection
Response accepting the request
Initial Handshake
(over HTTP)
Data frame
Data frame
Data frame
Data frame
Bidirectional Communication
(over persistent
WebSocket connection)
Connection close request
Connection close response
Closing Connection

REST	WebSockets
A uni-directional protocol	A bi-directional protocol
Follows stateless communication strategy for incoming HTTP payloads	Follows stateful communication strategy by maintaining a session state for each client
Uses a synchronous request-response model	Uses an asynchronous full-duplex messaging model
Uses TCP protocol as the underlying transport layer, and always uses the HTTP application layer protocol	Uses TCP protocol as the underlying transport layer, and uses the HTTP application layer protocol for the initial handshake, then upgrades the connection to the WebSocket protocol
Needs workarounds like polling to implement bi-directional communication channels	Developers can easily make the connection uni-directional (request-responsed-based) by using unique message identifiers

Thank You